OpenGL ES Overview for iPhone OS

Session 311

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

OpenGL ES Overview for iPhone OS

Michael Swift and Alex Kan
Software Engineers
Embedded Graphics Acceleration

Agenda

- Overview of OpenGL ES
  - OpenGL ES 1.1 and OpenGL ES 2.0
- Creating an OpenGL ES application
- Compositing with Core Animation and UIKit
- Device specifics
- Q&A

Agenda

Additional OpenGL ES sessions

OpenGL ES Shading and Rendering	Presidio Wednesday 3:30PM

- Shaders and programmability
- Advanced 3D rendering
- Image processing

OpenGL ES Tuning and Techniques	Presidio Wednesday 5:00PM

- Identifying and optimizing pipeline bottlenecks
- Optimizing shaders

Agenda

- Overview of OpenGL ES
  - OpenGL ES 1.1 and OpenGL ES 2.0
- Creating an OpenGL ES application
- Compositing with Core Animation and UIKit
- Device specifics
- Q&A

OpenGL ES API

What is OpenGL ES?

- Open industry standard
- Derived from OpenGL
- Streamlined 3D API for embedded systems
  - Simplified, nonredundant API
  - Small memory and CPU footprint
- Defines the 3D graphics pipeline

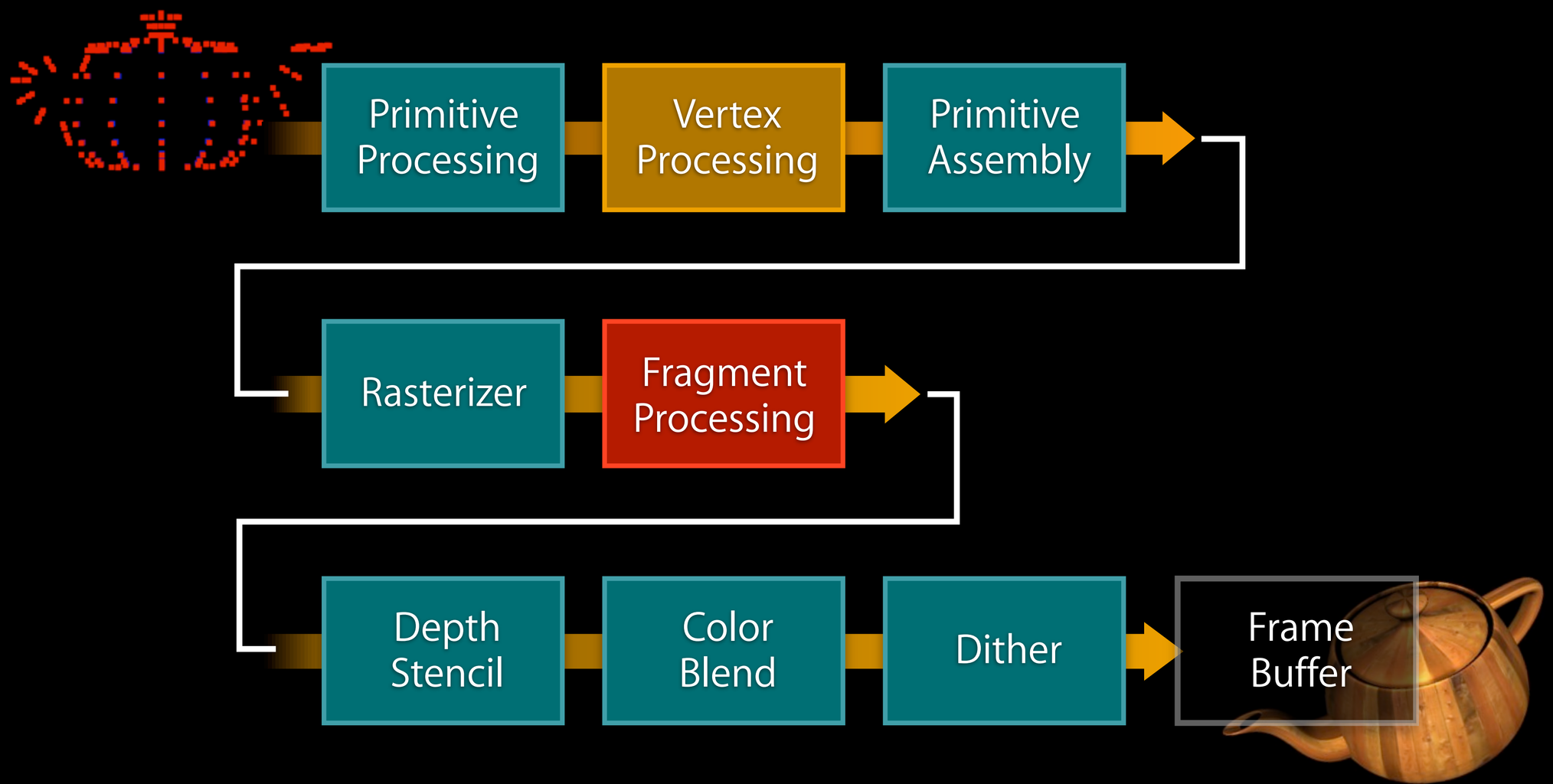
OpenGL ES Graphics Pipeline

OpenGL ES 1.1 vs. OpenGL ES 2.0

- Pipeline architecture is common
  - Most of the pipeline stages are identical
  - Vertex and Fragment stages are different
- OpenGL ES 1.1 implements a fixed-function pipeline
- OpenGL ES 2.0 implements a programmable pipeline

OpenGL ES Graphics Pipeline

Fixed-function graphics pipeline

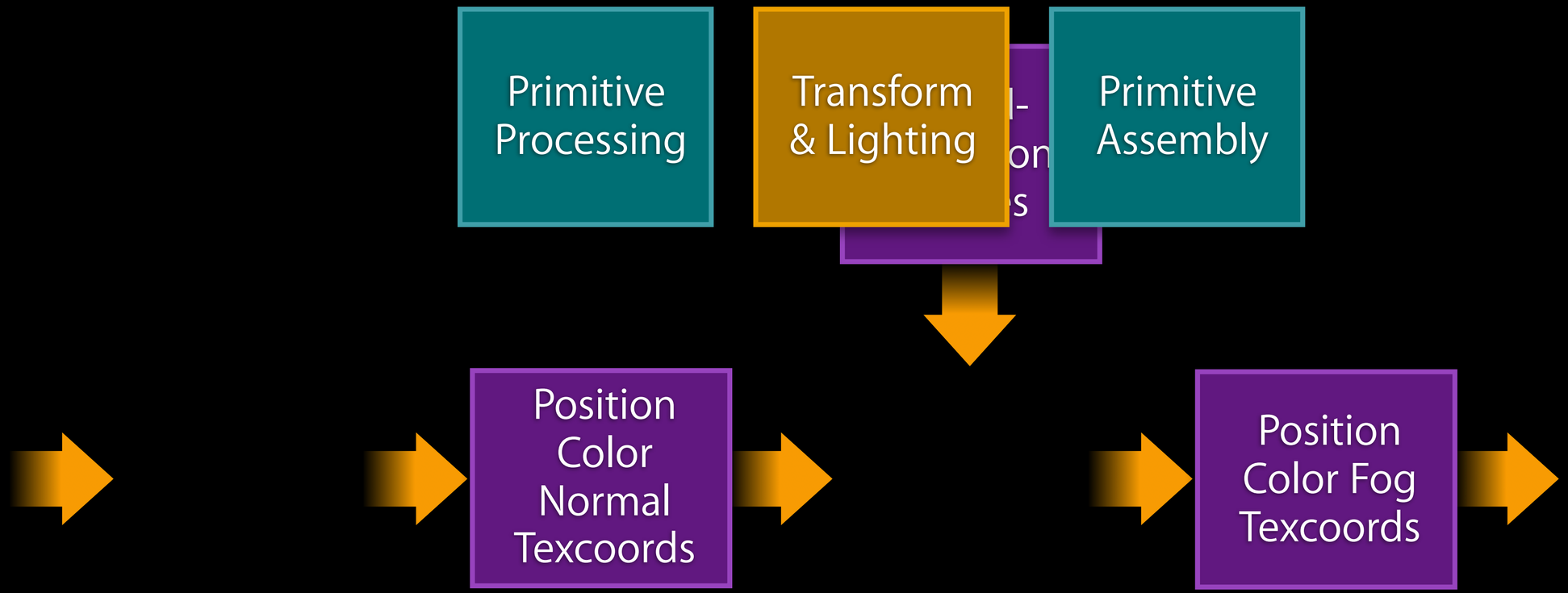


OpenGL ES 1.1

Fixed-function vertex processing

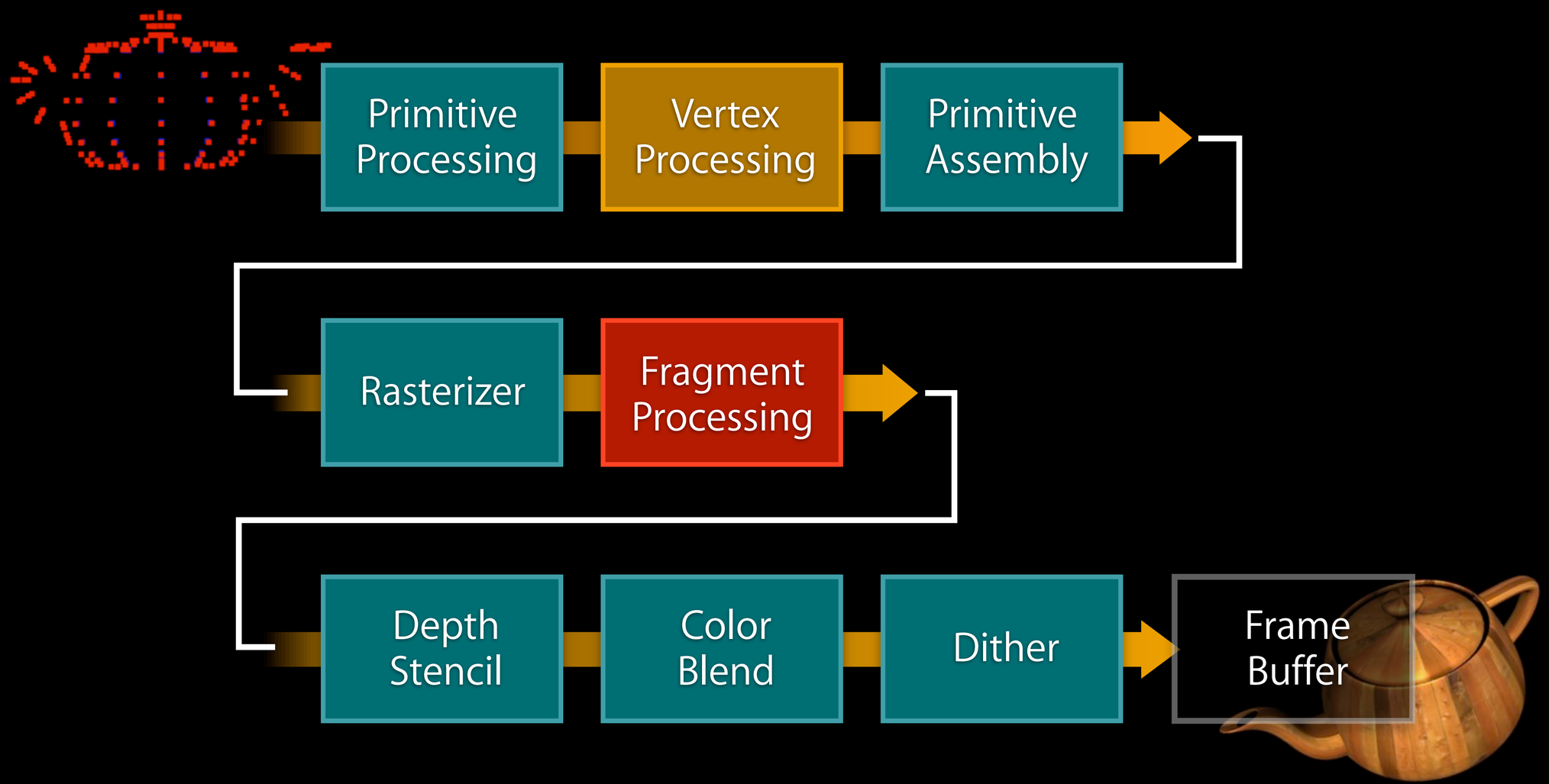
OpenGL ES 1.1

Fixed-function vertex processing



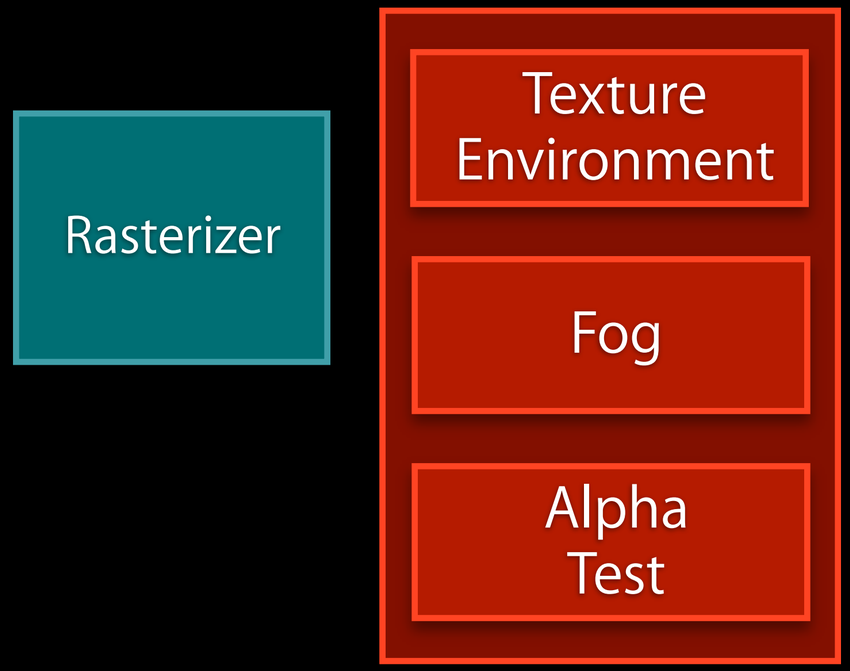
OpenGL ES Graphics Pipeline

Fixed-function graphics pipeline



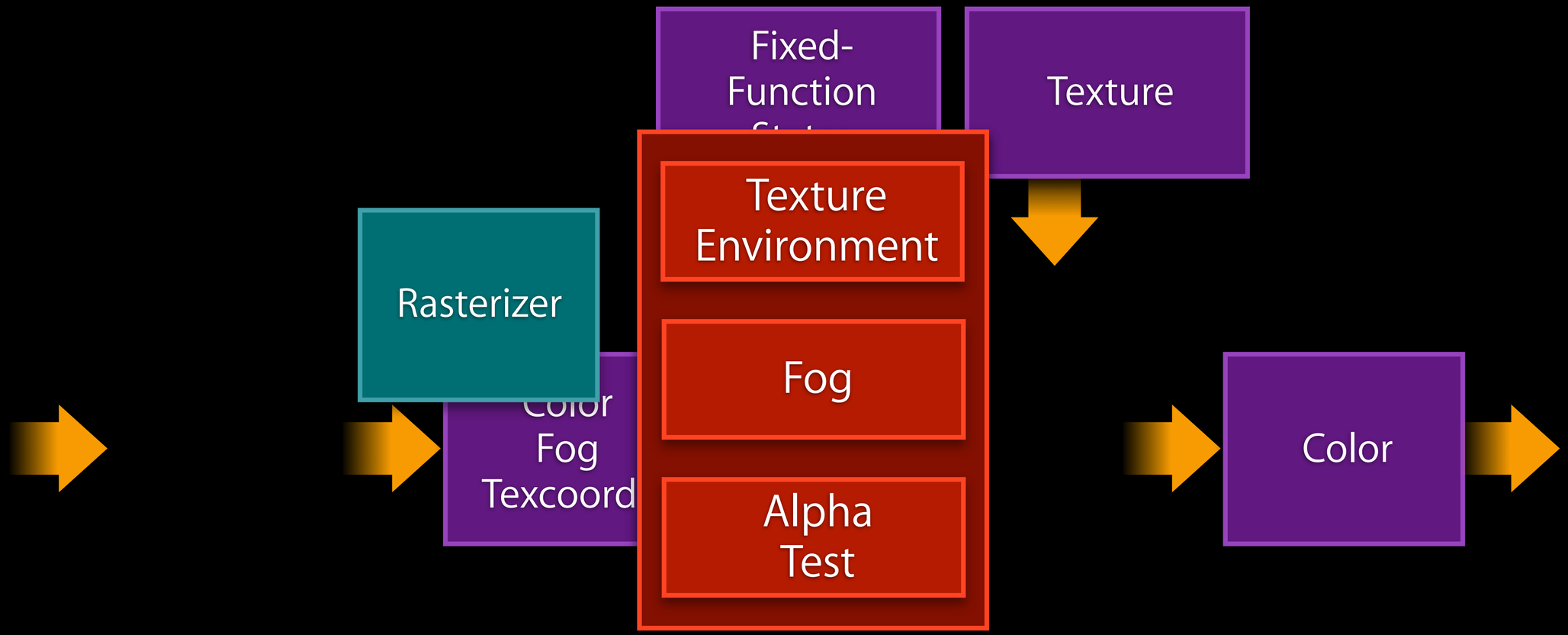
OpenGL ES 1.1

Fixed-function fragment processing



OpenGL ES 1.1

Fixed-function fragment processing



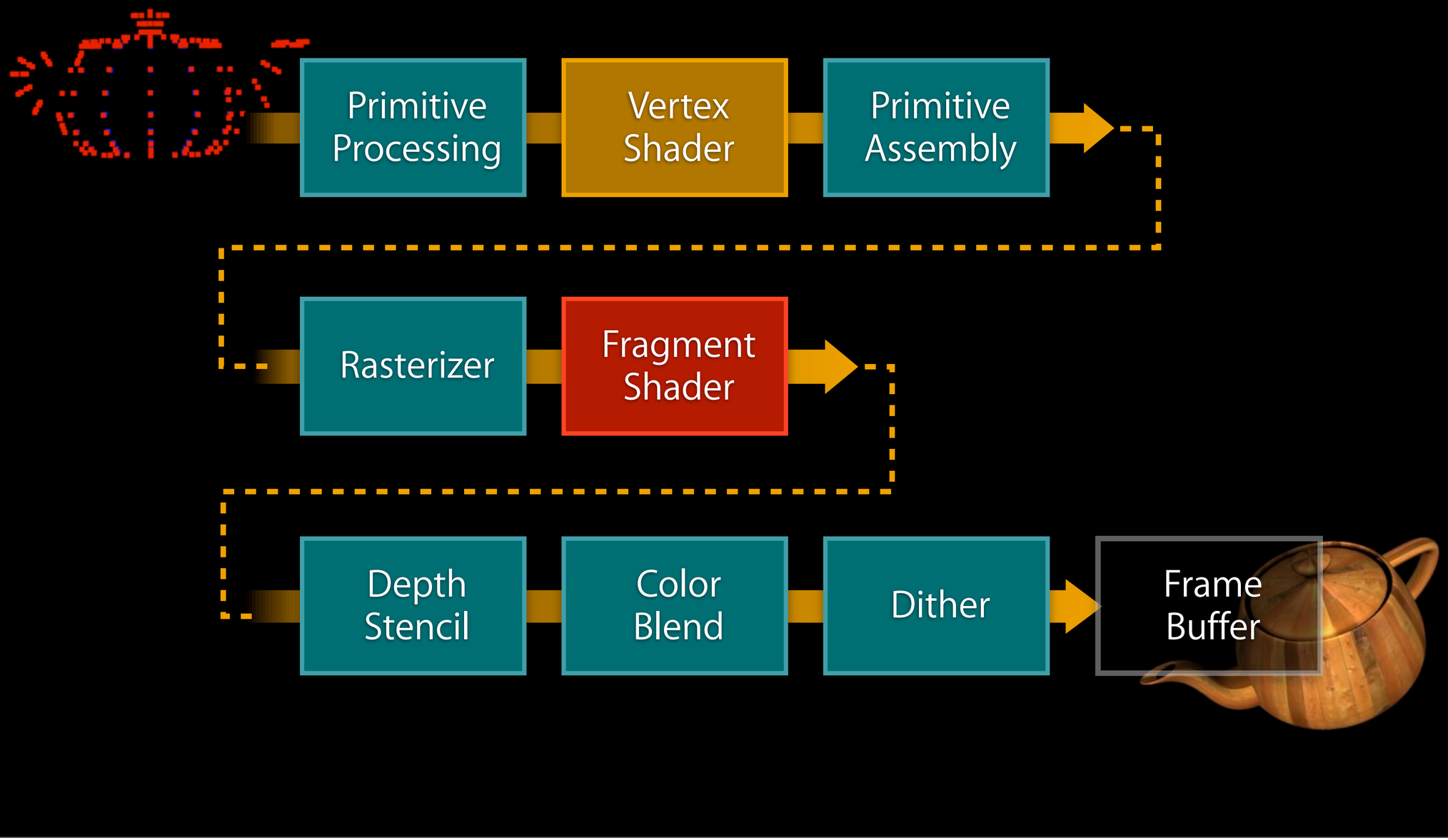
OpenGL ES 1.1

Fixed-function pipeline

- Vertex stage performs transform and lighting
- Fragment stage performs texturing, fog, alpha test
- Configured through enables, parameters, etc.
- API provides methods for managing fixed-function state
  - Example: Matrix transforms and stacks

OpenGL ES Graphics Pipeline

Programmable graphics pipeline



OpenGL ES 2.0

Vertex and fragment shaders

```
void main(void)
{
  gl_Position = ...
}
```

Vertex Shader

```
void main(void)
{
  gl_FragColor = ...
}
```

Fragment Shader

Program

OpenGL ES 2.0

What is a shader?

- Application code running in a pipeline stage
  - Performs work on a single vertex or fragment
- Written in a high-level, C-like language
- Specifies work, inputs and outputs, and parameters
- Naming of inputs and outputs depends on stage

OpenGL ES 2.0

Programmable graphics pipeline

- Vertex shader must output position
- Fragment shader must output color or discard
- Pipeline configured by vertex and fragment pairings
- API manages programs and remaining fixed-function blocks
  - Programmable: Shaders, program state, inputs, etc.
  - Fixed-function: Depth, stencil, blending, etc.

Choosing an OpenGL ES API

Three reasons for ES 1.1 vs. ES 2.0

- Programmability
- Hardware support
- Ease-of-use

Choosing an OpenGL ES API

Programmability

- OpenGL ES 1.1
  - Limited by texture environment math and texture units
  - Complex operations usually require multipass or multidraw
- OpenGL ES 2.0
  - Numerical range, flexibility
    - Unrestricted by fixed-function hardware
  - Can perform many ES 1.1 multidraw operations in a single draw
    - Generally faster and easier to understand
  - Can pass any data into OpenGL ES 2.0

Choosing an OpenGL ES API

Hardware support

- OpenGL ES 1.1 is supported on all devices
- iPhone 3G S adds OpenGL ES 2.0 support

Choosing an OpenGL ES API

Ease-of-use

- OpenGL ES 1.1
  - Requires little setup to draw
  - Has many APIs to group and manage state and parameters
    - Examples: Matrix transforms, state enables, push/pop semantics
- OpenGL ES 2.0
  - Requires more setup than OpenGL ES 1.1 to draw
    - Application must define the work it wants done
  - State management is done by the application
    - Examples: Write your own matrix operations and program selection

Choosing an OpenGL ES API

Summary

- Programmability
- Hardware support
- Ease-of-use

OpenGL ES Overview

Summary

- OpenGL ES 1.1 and 2.0 share a common pipeline
  - Only vertex and fragment stages differ
- OpenGL ES 1.1 is a fixed-function pipeline
- OpenGL ES 2.0 is a programmable pipeline

Agenda

- Overview of OpenGL ES
  - OpenGL ES 1.1 and OpenGL ES 2.0
- Creating an OpenGL ES application
- Compositing with Core Animation and UIKit
- Device specifics
- Q&A

Setting up an OpenGL ES Application

What do you need?

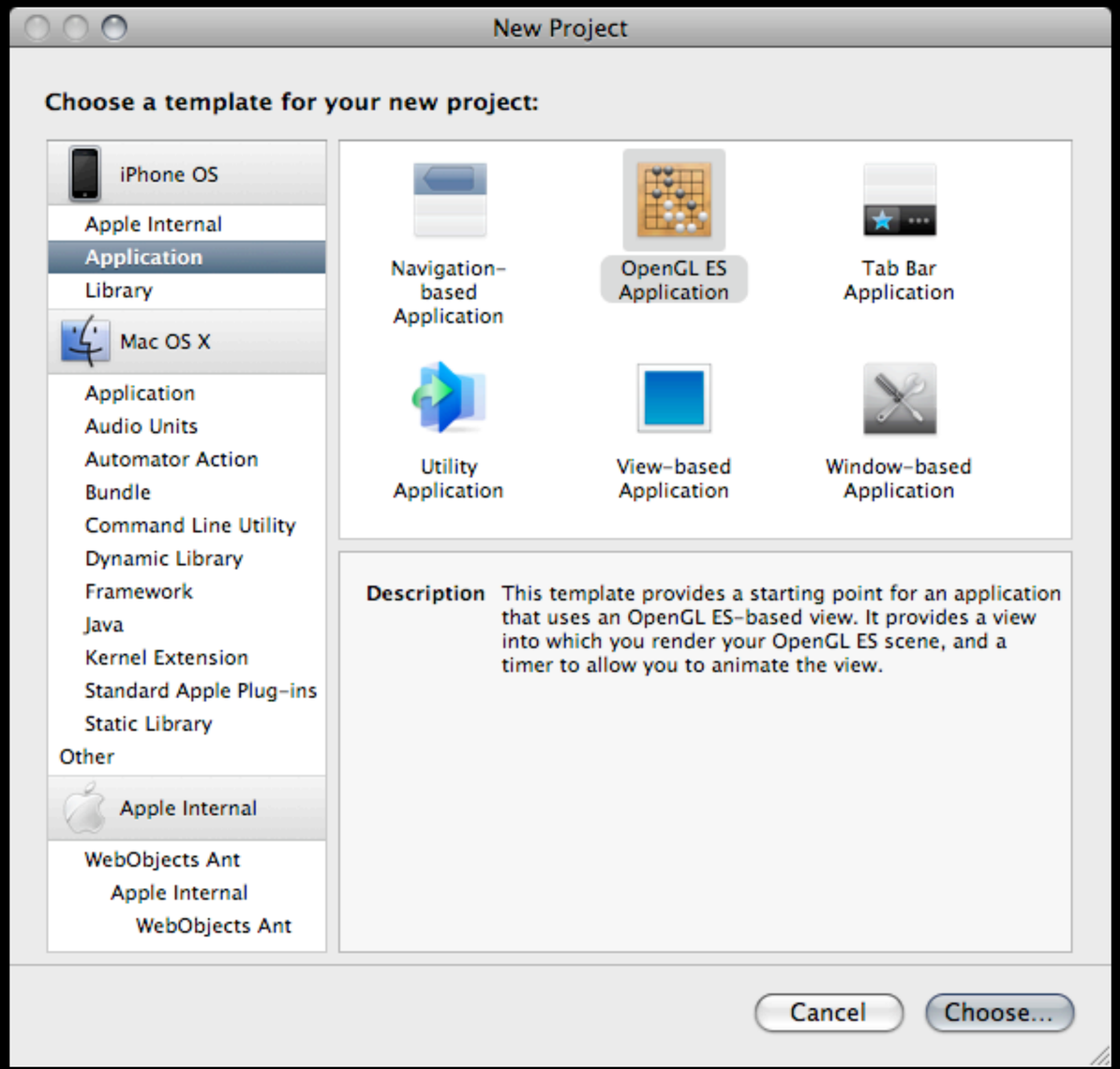
- Displayable rendering destination
- OpenGL ES context

Creating an OpenGL ES Application

Where do you start?

- Use the OpenGL ES template
  - Supports OpenGL ES 1.1
  - Support for OpenGL ES 2.0 can be added
  - Ready to go
- Understand how the OpenGL ES template works
  - Set up OpenGL ES
  - Drawing into a UIView

Using the OpenGL ES Template



Using the OpenGL ES Template

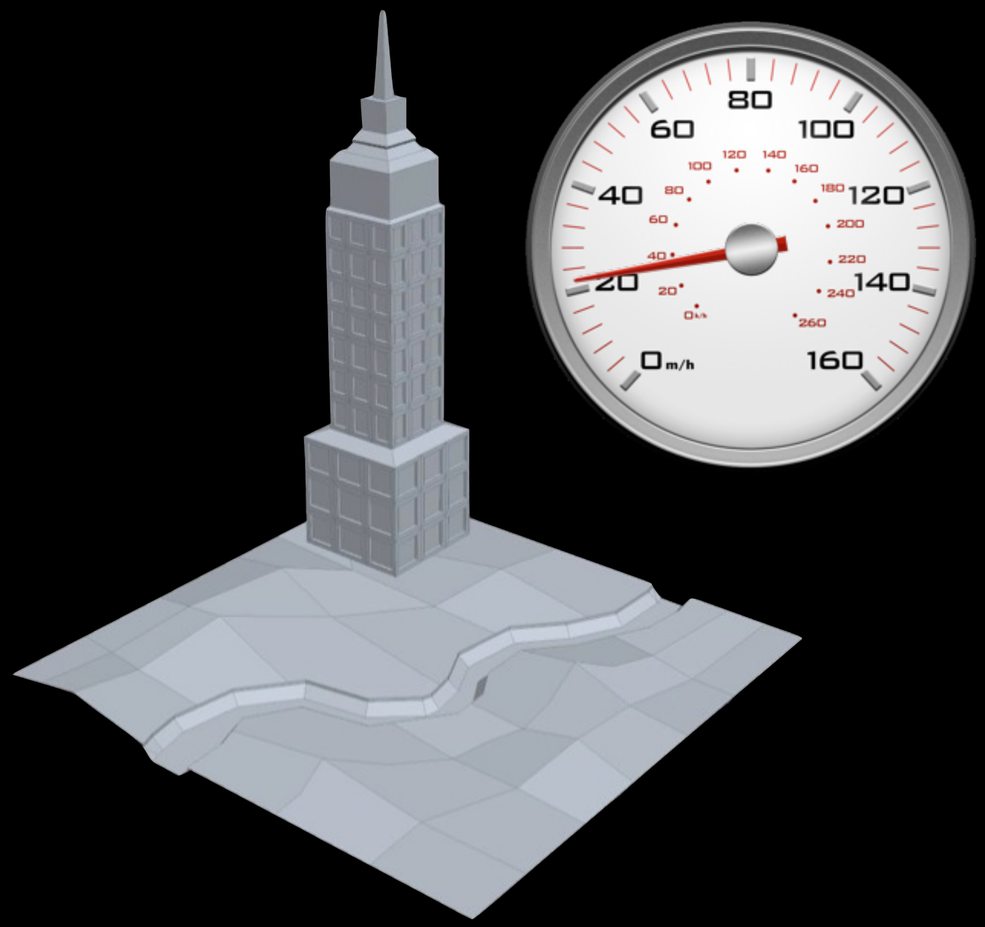
What does the template set up?

- Displayable buffer provided by EAGLView
- Configures OpenGL ES
  - Chooses OpenGL ES 1.1 API
  - Rendering goes into displayable buffer
- Drawing callback

Using the OpenGL ES Template

What are the next steps?

- Provide content
  - Textures
  - Models
  - Drawing code
- Customize UIKit interaction
  - Choose a framerate
  - Handle touch and accelerometer input
  - Respond to UIApplication events



Demo

Customizing the OpenGL ES template

Using the OpenGL ES Template

Summary

- Easy-to-use
  - Just add content and draw
- Targets OpenGL ES 1.1

Creating an OpenGL ES Application

Where do you start?

- Use the OpenGL ES template
  - Supports OpenGL ES 1.1
  - Support for OpenGL ES 2.0 can be added
  - Ready to go
- Understand how the OpenGL ES template works
  - Set up OpenGL ES
  - Drawing into a UIView

Creating an OpenGL ES Application

How does iPhone OS display content?

CALayer

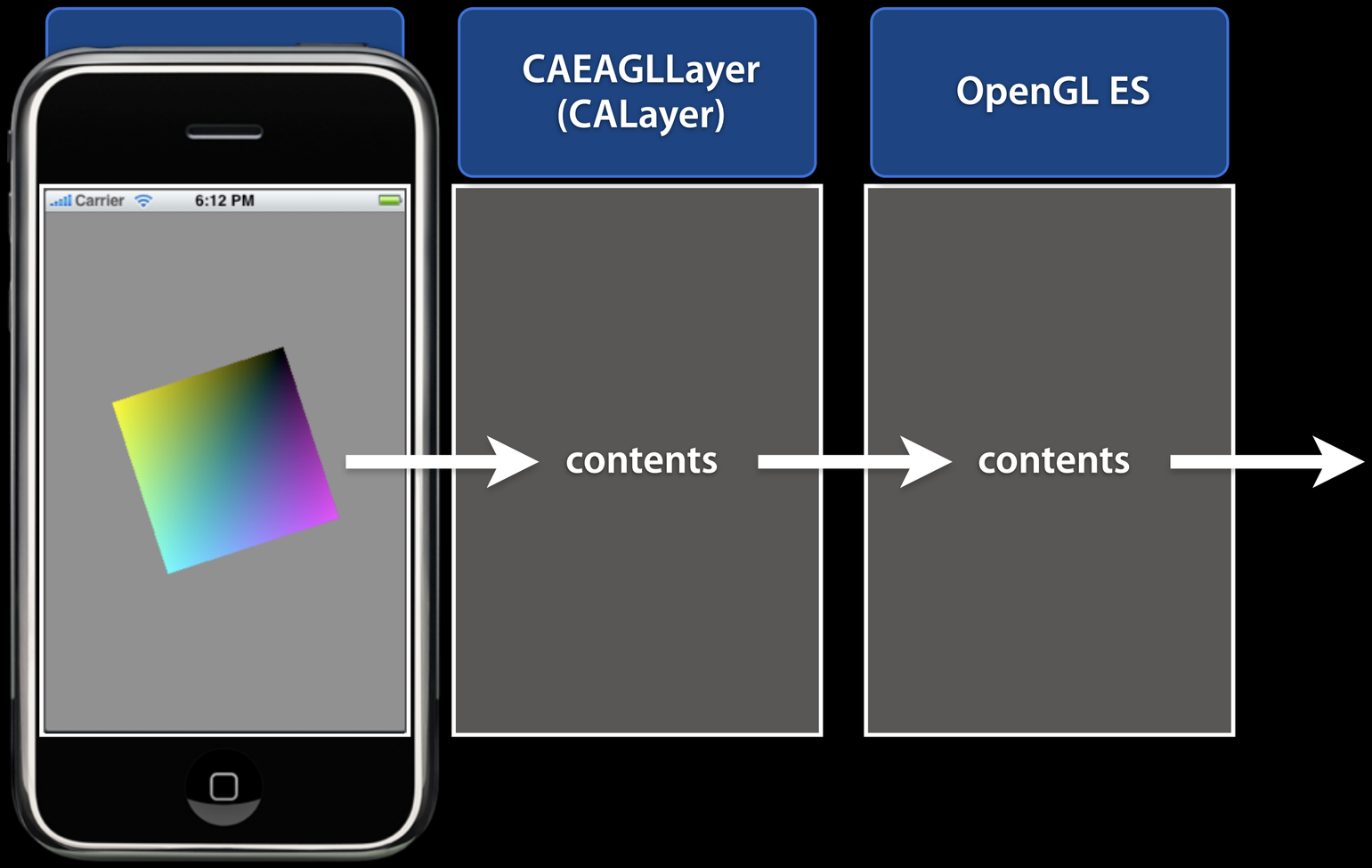
CGImage

contents

contents

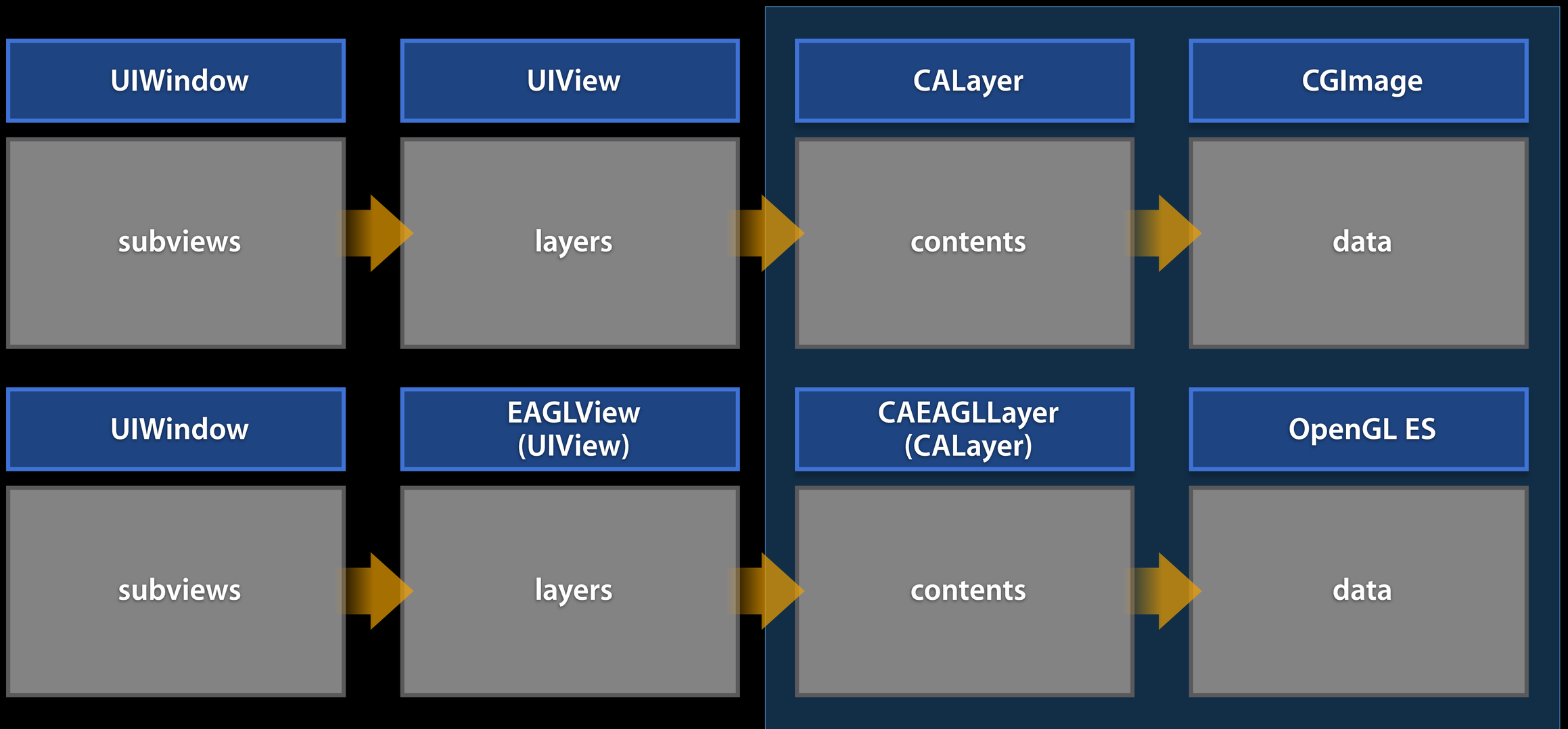
Creating an OpenGL ES Application

How does iPhone OS display OpenGL ES content?



Creating an OpenGL ES Application

Comparing UIKit and OpenGL ES



Setting up an OpenGL ES Application

Five steps

- Customize UIView
- Initialize OpenGL ES
- Set up OpenGL ES framebuffer
- Draw
- Present

Step 1: Customizing UIView

Enabling the UIView to support OpenGL ES

- Specify UIView's layerClass as CAEAGLLayer
  - Indicates that contents come from OpenGL ES

```
@implementation MyCustomView

/* Request CAEAGLLayer instead of CALayer */
+ (Class)layerClass {
    return [CAEAGLLayer class];
}

@end
```

Step 1: Customizing UIView

What is CAEAGLLayer?

- Displayable color destination for OpenGL ES rendering
- Optionally configured key-value pairs in drawableProperties
  - Color buffer format (kEAGLDrawablePropertyColorFormat)
    - kEAGLColorFormatRGB565
    - kEAGLColorFormatRGBA8
  - Retained backing store (kEAGLDrawablePropertyRetainedBacking)
    - [NSNumber numberWithBool:value] (YES or NO)

Step 2: Initializing OpenGL ES

What is an EAGLContext?

- Contains OpenGL ES state and command stream
- Selects OpenGL ES API version at creation
- Has no default rendering destination or configuration
- API calls issued to current EAGL context

Step 2: Initializing OpenGL ES

Setting up an ES 1.1 context

```
/* Allocate and initialize the EAGL Context */
EAGLContext* myCtx =
 [[EAGLContext alloc]
 initWithAPI:kEAGLRenderingAPIOpenGLES1];
```

```
/* Make this context current for the thread */
[EAGLContext setCurrentContext:myCtx];
```

```
/* You can now issue OpenGL ES commands */
const char* myRenderer = glGetString(GL_RENDERER);
```

Step 3: Framebuffer Setup

Configuring where OpenGL ES renders

- OpenGL ES renders into renderbuffers (RBs) and textures
- Framebuffer objects (FBOs) group renderbuffers for drawing
  - Can reference color, depth, and stencil renderbuffers
- `OES_framebuffer_object` API configures rendering destinations and groupings

Step 3: Framebuffer Setup

What is a renderbuffer object?

- Used by OpenGL ES as the destination for rendering
- Defines the storage of the data
  - Format
    - Examples: GL_RGBA, GL_DEPTH_COMPONENT16_OES
  - Allocation
    - Width and height

Step 3: Framebuffer Setup

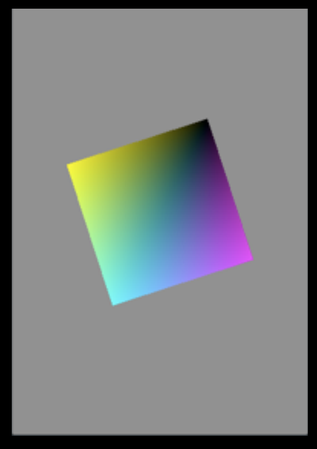
What is a framebuffer buffer object?

- Determines where OpenGL ES renders
- Grouping of renderbuffers
- Supports color, depth, and stencil attachments
- Must test for valid permutation of attachments

Step 3: Framebuffer Setup

FBOs, Renderbuffers, and CAEAGLLayer

CAEAGLLayer



Color Renderbuffer

- glGenRenderbuffer
- glBindRenderbuffer
- [myCtx renderbufferStorage: fromDrawable:]

Depth Renderbuffer

- glGenRenderbuffer
- glBindRenderbuffer
- glRenderbufferStorage

Framebuffer object

- glGenFramebuffer
- glBindFramebuffer
- glFramebufferRenderbuffer
- glFramebufferRenderbuffer
- glCheckFramebufferStatus

Step 3: Framebuffer Setup

Supported configurations

- iPhone, iPhone 3G, iPod touch (PowerVR MBX Lite)
  - Supports color and depth
- iPhone 3G S (PowerVR SGX)
  - Supports color, depth, and stencil

Step 4: Drawing

Render your content

Step 5: Presenting

Display your OpenGL ES content

- Update the contents of the UIView
  - Bind the CAEAGLLayer-backed color renderbuffer
  - Present the current renderbuffer

OpenGL ES Setup Summary

Five steps

- Customize UIView
- Initialize OpenGL ES
- Set up OpenGL ES framebuffer
- Draw
- Present

OpenGL ES Setup

How is ES 2.0 setup different?

- It’s the same process
- OpenGL ES 2.0 is not supported on all devices
  - Check for context creation success

Initializing OpenGL ES

Setting up an ES 2.0 context

```
/* Allocate and initialize ES 2.0 */
EAGLContext* myCtx =
 [[EAGLContext alloc]
 initWithAPI:kEAGLRenderingAPIOpenGLES2];
```

```
/* Fallback to ES 1.1 */
if(myCtx == NULL) {
 myCtx =
  [[EAGLContext alloc]
  initWithAPI:kEAGLRenderingAPIOpenGLES1];
}
```

Demo

Building an ES 1.1 and ES 2.0 compatible app

Agenda

- Overview of OpenGL ES
  - OpenGL ES 1.1 and OpenGL ES 2.0
- Creating an OpenGL ES application
- Compositing with Core Animation and UIKit
- Device specifics
- Q&A

Compositing with Core Animation and UIKit

OpenGL ES and UIView

- What's the same about using CAEAGLLayer?
  - View contents persist until next update
  - Regular UIView properties apply
  - CA uses the same compositing model
- What's different?
  - CAEAGLLayer contents are opaque by default

Compositing with Core Animation and UIKit

Compositing

- OpenGL ES–rendered views are first-class citizens in view hierarchy
  - All window system compositing operations just work
  - Some compositing operations are faster than others

Compositing with Core Animation and UIKit

View positioning for faster compositing

- OpenGL ES views composite faster if
  - No transforms are applied
  - Frame rectangle is aligned to pixel boundaries

Compositing with Core Animation and UIKit

Blended content and faster compositing

- OpenGL ES views composite faster if they are fully opaque
  - Contents are opaque
  - View opacity is 1.0
- OpenGL ES views can be obscured by other UIKit content
  - Examples: System keyboard, alert views, sheets
- If you need to blend OpenGL ES content, it must be premultiplied

Compositing with Core Animation and UIKit

Faster compositing with view effects

- Transitions and animations draw faster when layer contents are static
  - Pause rendering during transition for better performance

Demo

Compositing UIKit and OpenGL ES content

3G
9:42 AM

Compositing with Core Animation and UIKit

Summary

- It just works
- For best performance, ensure your OpenGL ES views are:
  - Screen-aligned
  - Opaque

Agenda

- Overview of OpenGL ES
  - OpenGL ES 1.1 and OpenGL ES 2.0
- Creating an OpenGL ES application
- Compositing with Core Animation and UIKit
- Device specifics
- Q&A

Device Specifics

OpenGL ES API support

- OpenGL ES 1.1
  - Supported on all devices and Simulator
- OpenGL ES 2.0
  - Supported on iPhone 3G S and Simulator

Device Specifics

Requiring OpenGL ES 2.0

- Edit application Info.plist

Key	Value
▼ Information Property List	(13 items)
Localization native development re	English
Bundle display name	${PRODUCT_NAME}
Executable file	${EXECUTABLE_NAME}
Icon file	
Bundle identifier	com.yourcompany.${PRODUCT_NAME:rfc1034identifier}
InfoDictionary version	6.0
Bundle name	${PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type code	????
Bundle version	1.0
Application requires iPhone enviror	☑
Main nib file base name	MainWindow
▼ Required device capabilities	(1 item)
Item 1	opengles-2

Device Specifics

OpenGL ES extensions

- Extensions allow implementations to provide additional functionality
- Must query presence of extension to use functionality
- Can vary by device and selected API

Device Specifics

OpenGL ES 1.1 extensions

- GL_EXT_texture_filter_anisotropic
- GL_EXT_texture_lod_bias
- GL_IMG_read_format
- GL_IMG_texture_compression_pvrtc
- GL_IMG_texture_format_BGRA8888
- GL_OES_blend_subtract
- GL_OES_compressed_paletted_texture
- GL_OES_depth24
- GL_OES_draw_texture
- GL_OES_framebuffer_object
- GL_OES_mapbuffer
- GL_OES_matrix_palette
- GL_OES_point_size_array
- GL_OES_point_sprite
- GL_OES_rgb8_rgba8
- GL_OES_texture_mirrored_repeat
- GL_APPLE_texture_2D_limited_npot
- GL_OES_packed_depth_stencil
- GL_OES_stencil8

Device Specifics

OpenGL ES 2.0 extensions

- GL_IMG_read_format
- GL_IMG_texture_compression_pvrtc
- GL_IMG_texture_format_BGRA8888
- GL_OES_depth24
- GL_OES_mapbuffer
- GL_OES_rgb8_rgba8
- GL_OES_standard_derivatives
- GL_OES_packed_depth_stencil

Device Specifics

OpenGL ES limits

- Capabilities are expanded when using iPhone 3G S (PowerVR SGX)
  - ES 1.1: Additional texture units and increased texture and drawable sizes
  - ES 2.0: Shaders and programmability
- Must query limits to ensure correct operation

Device Specifics

Summary

- Don't assume
  - Extensions
  - Implementation limits

Contact Info

Allan Schaffer
Graphics Technology Evangelist
aschaffer@apple.com

Related Sessions

OpenGL ES Shading and Rendering	Presidio Wednesday 3:30PM
OpenGL ES Tuning and Techniques	Presidio Wednesday 5:00PM

Labs

OpenGL ES Lab	Graphics and Media Lab A Thursday 12:00PM
Game Development for iPhone	iPhone Lab C Thursday 9:00AM

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09